



① Publication number : **0 499 394 A1**

⑫ **EUROPEAN PATENT APPLICATION**

②① Application number : **92300919.5**

⑤① Int. Cl.<sup>5</sup> : **G06F 13/38**

②② Date of filing : **04.02.92**

③① Priority : **12.02.91 US 654441**

④③ Date of publication of application :  
**19.08.92 Bulletin 92/34**

⑧④ Designated Contracting States :  
**DE FR GB IT**

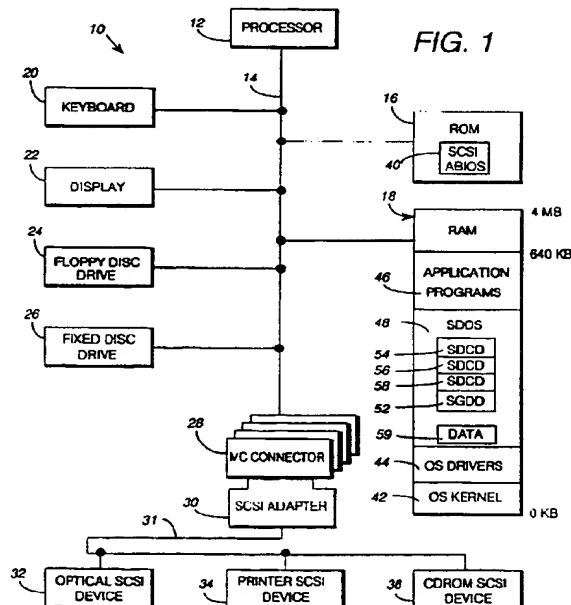
⑦① Applicant : **International Business Machines Corporation**  
**Old Orchard Road**  
**Armonk, N.Y. 10504 (US)**

⑦② Inventor : **Feriozi, Dan Trotter**  
**1431 S. Ocean Boulevard No. 20**  
**Pompano Beach, Florida 33062 (US)**  
Inventor : **Khatri, Bharat**  
**4283 Brandywine Drive**  
**Boca Raton, Florida 33487 (US)**

⑦④ Representative : **Blakemore, Frederick Norman**  
**IBM United Kingdom Limited Intellectual Property Department Hursley Park**  
**Winchester Hampshire SO21 2JN (GB)**

⑤④ **Device drivers for multitasking operating system.**

⑤⑦ A tiered device driver system includes a SCSI generic device driver (SGDD) in one tier and one or more SCSI device-class drivers (SDCD) in another tier. In response to a request to access a SCSI device, the operating system creates a request packet that is passed to the appropriate SDCC. Such SDCC creates a generic request packet and associated data structures that contain information specific to the SCSI device being accessed. The generic request packet is passed on to a SCSI generic device driver (SGDD) that creates a SCSI ABIOS request block which it transmits to a SCSI adapter for accessing the desired SCSI device. The SGDD provides functions common to the SDCCs.



EP 0 499 394 A1

This invention relates to the field of data processing, and, more particularly to improvements in device drivers for controlling or managing the flow of data to and from devices in a programmed data processing system having a multitasking operating system such as OS/2 operating system.

## 5 BACKGROUND OF THE INVENTION

By way of background, device drivers are programs or routines which control or manage the flow of data to and from I/O devices. The drivers form part of and interact with other portions of an operating system. An operating system normally includes a basic set of device drivers for I/O devices, such as a keyboard, fixed and floppy disks, display, and printer, commonly used in a personal computer. When an I/O device is added to a data processing system, and such device is not operable under an existing driver, a new driver must be added to the system in order to use the device. Such new driver is customarily supplied by the maker of the I/O device and is installed in the system in accordance with procedures established by the operating system. In personal computers operating with IBM DOS or OS/2 operating systems, such drivers are installed, when the computers are started or rebooted, using commands or instructions in a CONFIG.SYS file.

The high performance models of the IBM PS/2 personal computers include a bus designed in accordance with Micro Channel architecture. (IBM, OS/2, PS/2 and Micro Channel are trademarks of International Business Machines Corporation). Such bus is referred to hereinafter as an "MC bus" and provides the means by which additional I/O devices and subsystems can be connected to the personal computers. A SCSI (Small Computer System Interface) bus is a bus designed in accordance with a particular architecture known as SCSI architecture, and provides a standardised design for the attachment thereto of I/O devices known as SCSI devices, that is, devices specifically designed for attachment to a SCSI bus. Such architecture defines a SCSI command set for accessing the devices. Recently, a SCSI adapter and SCSI ABIOS (advanced basic input/output operating system) were developed which allow SCSI devices to be connected to PS/2 computers through the MC bus, and this has created some difficulties or problems.

First, the basic OS/2 drivers for common (non-SCSI) I/O devices cannot be used with SCSI devices, and a driver written for use with one type of operating system cannot be used with another type. Accordingly, the makers of SCSI devices are faced with the prospect of having to write multiple drivers, one for each type of operating system for which they expect to market a SCSI device. A complete driver is relatively complex and commonly requires many programmer months to develop. This can add up to a substantial development effort if several devices must be supported under different operating systems, and it may delay the general availability and widespread use of SCSI devices. The invention, as described in greater detail below, has an objective of simplifying the driver to be provided by a maker or supplier of SCSI devices for use in a data processing system programmed to operate under OS/2 operating system. Simplification makes it easier and cheaper to develop and supply such drivers. This objective is accomplished by a two tier driver system in which the SCSI developer provides a SCSI driver that is specific to a class of SCSI devices, and the operating system includes a generic SCSI driver having functions commonly used by the specific driver classes. Not only does such system eliminate the need for developers to include common functions but it also hides from the developer the need to program to the relatively complex interface with ABIOS.

Second, PS/2 personal computers include microprocessors that operate in both real mode and protected mode. Bimodal operation provides compatibility with older application programs and allows such microprocessors to run both application programs written for a DOS environment and application programs written for an OS/2 environment. Device drivers are also bimodal so that such drivers and corresponding devices can be used for both DOS and OS/2 application programs. Under OS/2, device drivers are loaded into the low end of physical memory for access by both DOS and OS/2 application programs. The more drivers there are, the more memory space is used thereby limiting the amount of physical memory space used for DOS application programs which run in real mode. Thus, another objective of the invention is to provide a SCSI driver system which efficiently uses memory address space. This objective is also satisfied by having a two tier driver system in which common functions are included in a generic driver. The common functions are used by SCSI device-class drivers and thus avoid the need for each driver to be complete in itself and thereby duplicate functions and waste memory space.

According to the present invention there is provided data processing system comprising at least one SCSI device of a predetermined SCSI device-class; a SCSI adapter connected to said SCSI device, said adapter being operable upon receiving a subsystem control block (SCB) to access said device, said SCB containing information affording access to said device; a read only memory for storing a SCSI advanced basic I/O system (ABIOS) program, said SCSI ABIOS program being operable to control access to said device in response to receiving an ABIOS request block, which includes said SCB; a main memory for storing a multitasking operating system, application program and data structures; and a processor for executing application programs and

operating said data processing system under control of said operating system; characterised in that the data processing system further comprising a device driver system comprising: a SCSI device class driver (SDCD) stored in said main memory, said SDCC being callable by said operating system and operable in response to receiving a first request packet from said operating system to construct a second request packet and said SCB; and  
 5 a SCSI generic device driver (SGDD) stored in said main memory, said SGDD being callable by said SDCC and operable upon receiving said second request packet from said SDCC to construct said BIOS request block and pass said BIOS request block to said SCSI BIOS, said second request packet containing information for passing said SCB to said SCSI BIOS

In the drawings:-

10 Fig. 1 is a block diagram of a data processing system embodying the invention;

Fig. 2 is a block diagram of a more detailed portion of the invention;

Fig. 3 is a block diagram illustrating certain data structures used in the invention;

Fig. 4 is flow diagram of the operation of a SCSI device-class driver;

Figs. 5A and 5B form a flow diagram showing task time operation of the SCSI generic device driver; and

15 Fig. 6 is a flow diagram showing interrupt time operation of the SCSI generic device driver.

## DETAILED DESCRIPTION

Referring now to the drawings, and first to Fig. 1, there is shown a data processing system 10 operable under an operating system such as OS/2. System 10 comprises a processor 12 connected to a bus system 14 which interconnects other elements of system 10. The other elements include a ROM (read only memory) 16, a RAM (random access memory) 18, a keyboard 20, a display 22, a floppy disc drive 24, a fixed disc drive 26, and a plurality of MC (Micro Channel) connectors 28. A SCSI host adapter 30 is plugged into one of connectors 28 and is connected to a SCSI bus 31 and three different types of exemplary SCSI devices 32, 34, and 36.  
 25 The types of illustrated SCSI devices are an optical SCSI device 32, a printer SCSI device 34, and a CDROM SCSI device 36. ROM 16 stores SCSI BIOS (advanced basic I/O system) 40.

The SCSI devices, in accordance with the SCSI architecture thereof, respond to device commands embedded in control blocks that are sent to a device from SCSI adapter 30. SDDS 48 functions to deliver information for the control block to SCSI BIOS 40 which delivers the required control block to the device being accessed.  
 30 Adapter 30 requires more information than just a SCSI control block. It needs to know the memory storage address of data and the requested direction of data flow, in order to control the flow of data to and from the SCSI device.

RAM 18 may be of a size up to sixteen MB (megabytes) and is typically four MB as shown in Fig. 1. After system 10 is started and initialized, the kernel 42 of the OS/2 operating system is stored in RAM 18 at the low end of the address space along with the basic OS drivers 44 for controlling the standard devices such as the keyboard, display, floppy disc drive, and fixed disc drive. When running in real mode, application programs 46 are stored in the address space below 640 KB (kilobytes) in whatever space is available after the other programs have been loaded in such space. The amount of memory allocated to such application programs is thus dependent on the sizes of such other programs including the device drivers. In order to operate the SCSI devices 32, 34, and 36, a SDDS (SCSI device driver system) 48 is also stored in the low end of RAM 18 for access by programs running in real and protected modes. SDDS 48 includes a two tiered driver system. One tier includes a SGDD (SCSI generic device driver) 52. The other tier includes a plurality of SDCC (SCSI device-class drivers) 54, 56, and 58 correspondingly specific to different classes of SCSI devices 32, 34, and 36. A given SDCC can access more than one device provided the device is of the class for which the driver is designed to operate.  
 40 SDDS also includes a data area 59 used by the programs. Except for SDDS 48, the remaining hardware and software of data processing system 10 are known, commercially available items.

Fig. 2 shows certain of the elements shown in Fig. 1, in a layered structure or model indicating the logical relationship of the elements, and their interfaces. Application programs 18 run in a real mode or a protected mode and coact with OS kernel 42 in order to use I/O devices including SCSI devices 32, 34, and 36. SDDS fits between kernel 42 and SCSI BIOS 40. Kernel 42 coacts with the tier of SDCCs 54, 56, and 58 through an interface 60 and also provides standard device driver helper routines known as DevHlp services. The SDCCs coact with SGDD 52 through an interface 62, and the SGDD coacts with SCSI BIOS 40, through an interface 64. SCSI BIOS 40 interacts with SCSI adapter 30 which in turn controls the various SCSI devices 32, 34, and 36. The interfaces between the elements, other than interfaces 60, 62, and 64, are known interfaces and need not be described in detail. The system is considered layered with application programs being the uppermost layer and hardware devices being the lowest layer. As control progresses through the layers from the top down, each layer becomes progressively more detailed and the SCSI BIOS layer is the software layer that handles the specific details of operating the specific SCSI devices. SCSI BIOS is designed to operate in a multitasking,  
 55

interrupt driven environment and is operating system independent. The interface 64 to SCSI ABIOS is complex and procedurally oriented. It hides the operating system details from ABIOS as well as hiding the details of the SCSI devices from the operating system.

Since the novelty in the invention results from the structure, functions and operation of SDDS 48, the interfaces used therewith will now be briefly described, with more detailed description and examples appearing below. Interface 60 is in accordance with the published interface described in IBM OS/2 Programming Tools and Information Version 1.2, I/O Subsystems and Device Support, Vol. 1- Device Drivers, Sept 1989. The principal data structure used in interface 60 is a conventional OS/2 request packet. Interface 64 is in accordance with the Supplement for the Personal System/2 and Personal Computer BIOS Interface Technical Reference, published December 1989 by IBM Corporation, Form number S15f-2161. The principal data structure used in interface 64 is a conventional ABIOS request block. Interface 62 is patterned after interface 60 and uses a generic IOCTL request packet.

The upper level driver, i.e., each one of the SDCDs, provides the direct connection of SDDS 48 to the operating system while the lower level driver (SGDD 52) provides access to the SCSI devices through services of SCSI ABIOS 40. Each SDCD is called a "device-class" driver because it generally drives that belong to a specific SCSI classification. The illustrated classes (printer, optical, and CDROM) are examples of SCSI device types that require a different upper-level class device driver to work in concert with the lower level SGDD. The primary functions of a device-class driver are to present a logical view of the device to the operating system, translate an OS/2 request packet into a SCSI control block and pass it on to the SGDD, and provide multiple vendor support by handling vendor unique features of the device. The general functions of SGDD 52 are to queue all SCSI requests by device, provide control block information to SCSI ABIOS, field all SCSI interrupts, and detect and handle timeout conditions.

As is publicly known, the SCSI architecture provides a standard device interface that greatly simplifies the firmware and software that is necessary to drive a SCSI device. SCSI devices have the ability to respond to commands embedded in control blocks. This means that a SCSI device driver's or driver system's primary task is simply to translate an operating system request block, which is created in response to an application program requesting access to a device, into a control block that the device can understand. The primary data structures involved in this process in accordance with the invention, are shown in Fig. 3. The arrows at the left of Fig. 3 suggest the overlapping nature of the data structures which are generally created by one layer but used by two adjacent layers. An application program initiates access to one of the SCSI devices, by making a system call to operating system kernel 42 which prepares an OS/2 request packet 66. The kernel then calls the appropriate SDCD, e.g. SDCD 54, which prepares a generic IOCTL request packet 68 using information in packet 66. Packet 68 includes a pointer 72 to a sense data buffer 74, and a pointer 76 to a parameter buffer 78. Parameter buffer 78 includes a pointer 80 to the header of a subsystem control block (SCB) chain 82 which is prepared by the SDCD and includes at least one SCB specific to the particular access. Other entries in the chain correspond to different device commands that result from the SDCD breaking down a more general OS/2 request into a series of more specific device commands. SCB chain 82 contains the information specific to the SCSI device being accessed. The SDCD then passes the location of packet 68 to the SGDD which prepares an ABIOS request block 70. Block 70 is then passed to SCSI ABIOS 40 which then sends information to adapter 30 to access the desired device. Upon completion of the device access, the data structures are accessed in the reverse sequence to pass information back to the application program. Thus, packets 66 and 68 and block 70 are the primary means of communication between OS/2, SDCDs, SGDD and SCSI ABIOS.

Request packet 68 contains fields logically divided into a static and variable sections. The fields contain the following information:

#### Static

Field	Information
1	Length of request packet
2	Block device unit code
3	Command code
4	Request packet status
5	Queue linkage

#### Variable

6	Function category
7	Function code
8	Parameter buffer address
9	Data buffer address

ABIOS request block 70 also contains static and variable parts the fields of which contain the following information:

**Static**

<b>Field</b>	<b>Information</b>
1	Request block length
2	Logical ID
3	Unit
4	Function
5	Return code (IN/OUT)
6	Time-out (OUT)

**Variable**

7	Physical pointer to SCB
8	Logical pointer to SCB chain header
9	Time to wait before resuming request (stage on time)
10	Flags
11	Status

SCB chain header 82 contains the following fields of information:

<b>Field</b>	<b>Information</b>
1	Pointer to next SCB chain header
2	Logical pointer to TSB
3	Command
4	Enable
5	Logical block address
6	System buffer address
7	System buffer byte count
8	TSB address
9	Optional SCB chain address
10	Block count
11	Block length

Field 1 and 2 form a header. Fields 3-11 form an SCB.

A termination status block (TSB) contains the following information:

<b>Field</b>	<b>Information</b>
1	SCB status
2	Retry counts
3	Residual buffer address
4	Residual buffer address
5	Address status length
6	Command/SCI status
7	Command/device error codes
8	Attachment diagnostic error modifier
9	Cache information
10	Last SCB address processed

Within data 59, SGDD 52 also maintains a SCSI device table for keeping track of what is going on, there being one entry for each SCSI device allocated. Each entry contains the following information:

<b>Field</b>	<b>Information</b>
1	Pointer to first waiting request
2	ABIOS logical ID
3	Interrupt level
4	Flags for indicating a busy device, staged-on interrupt, sense data needed, allocated device, removable media, retry occurred, request sense called, primary timeout occurred, and secondary timeout occurred.

Data 59 also includes a started queue which is a chain of ABIOS request blocks 70 that have already called the ABIOS start entry point. This queue is used by the interrupt handler to get ABIOS blocks for calling the ABIOS Interrupt entry point. Such queue allows SGDD to handle concurrent plural requests to access plural SCSI devices. Data 59 also includes a waiting queue for receiving ABIOS request blocks when a particular SCSI device is busy and there is more than one request to access such busy device.

The system is initialised in conventional fashion except that communication between the SDCDs and SGDD is through OS/2 architected inter-device driver communication (IDC) facility. At initialisation time, SGDD 52 registers its IDC entry point with the OS kernel 42. Later, when a SDCD initialises, it obtains such entry point from the kernel for use, as described below, in passing control to SGDD 52.

In response to an application request, kernel 42 calls the appropriate SDCD. Referring to Fig. 4, the OS kernel calls the SDCD at its strategy entry point with a request packet 66 in step 84. SDCD first validates and routes the command specified in such packet, in step 86. Then step 88 allocates memory space for and builds an SCB chain header 82 for the request. Step 90 then allocates memory for and builds a generic ioctl request packet 68. Next, in step 92, SDCD points es:bx to packet 68, sets up a context for the SGDD, makes a far call to the SGDD's IDC entry point, and waits for the return. When control is returned to SDCD, step 94 processes any error that may have occurred while accessing the desired device. Step 96 then posts data to the application caller if indicated or if it was not already done by a DMA controller from the SCSI adapter 30. Step 98 sets the status code and step 100 then returns to the caller through the OS kernel.

Referring to Fig. 5, the SGDD task time operation begins in step 102 when an IOCTL request packet 68 is passed to SGDD 52 by a far call from SDCD by step 92 (Fig. 3). SGDD has two entry points, an IDC entry 104 and a strategy entry 106 which are the same. Step 108 validates and routes the OS/2 command. Step 110 is entered by recognising that the OS command is a generic IOCTL command and step 110 validates and routes such command using the function category and function code information in packet 68. In step 112, memory is allocated for an ABIOS request block 70 which is then built by inserting information in its fields as appropriate. Step 114 decides if the device is busy. If not, step 120 puts the request block built in step 112 onto the started queue and calls ABIOS 40 through the Start entry point. If the device is busy, step 116 puts the request block on a waiting queue and blocks by calling the OS/2 DevHlp services. SGDD then waits for a return from ABIOS and the next step will depend on the return code from ABIOS.

Such return code will indicate different conditions, "done", "staged on interrupt", or "staged on time". If the return code indicates a "done" situation, control passes to step 125. If the return code indicates a "staged on interrupt", step 122 calls kernel 42 to perform a block on interrupt operation and upon completion the kernel will provide a return code indicating "run" or "timeout". If the result as defined by the return code from kernel 42 in step 122 is "run" step 125 is then performed. If the result is a "timeout" step 124 calls ABIOS timeout 124 after which step 125 is performed. If the result of step 120 is a "staged on time", step 126 then blocks for time and upon "wakeup", step 128 calls ABIOS interrupt.

Step 125 examines the ABIOS return code and determines if an error was encountered while attempting to access the device. If no error occurred, step 128 deallocates the ABIOS request block, step 130 posts the results of the command, and step 132 returns to the caller. If there was an error, step 136 calls an error handler and step 138 determines if sense data is needed. This is done by examining the associated TSB to determine the nature of the error. If the TSB indicates that more error information should be requested, a request sense command is sent to the device by step 142. The sense data error information is sent directly (step 144) from the device to the associated upper level SDCD through the sense data pointer that is part of the generic IOCTL request packet. The lower level driver (SGDD) then returns an error code indicating that valid sense data is available to the upper level driver to interpret. If sense data is not available from the device, the ABIOS error code is mapped by step 140 to an appropriate OS/2 device driver return code and returned to the upper level driver. The automatic requesting of sense data by the lower level driver saves time and code in the upper level drivers, as well as ensuring that sense data is not lost by a subsequent command execution that may have been queued in the lower level driver. The driver that queues commands must also issue timely requests for sense data. Following step 140, steps 128, 130 and 132 are sequentially performed. If a timeout 134 occurs while blocked in step 116, steps 128, 130, and 132 are performed.

Referring to Fig. 6, in step 152, the OS interrupt handler first receives the interrupt, determines it is for the SCSI drivers and then calls the SCSI interrupt handler 154 which then proceeds to perform the remaining steps shown in Fig. 6. Step 156 decides if there is any pending request on the Started Queue. If the started queue is not empty, a request is pending, i.e., staged on interrupt. If there is a request pending, step 158 calls the ABIOS interrupt entry point once for each request on the started queue, and provides the address of that pending ABIOS request block to ABIOS for service. If there is no request pending, step 162 calls the ABIOS interrupt entry point once for each device in the system. After either of steps 158 or 162, step 160 decides if there is a claimed interrupt, i.e., ABIOS indicates one of the SCSI devices caused the interrupt. If not, a branch is made to step 174. If there is a claimed interrupt, step 164 runs the pending block thread corresponding to the serviced request. If ABIOS indicates that an error has occurred, step 166 checks the associated TSB to see if such error requires a request sense call. If not, step 172 runs the first thread waiting for access to the device that caused the interrupt. If a request sense is needed or upon completion of step 172, step 170 issues an end of interrupt call to OS/2. Finally, step 174 returns to OS/2 and indicates whether or not an interrupt is claimed.

## Claims

1. A data processing system comprising at least one SCSI device of a predetermined SCSI device-class; a SCSI adapter connected to said SCSI device, said adapter being operable upon receiving a subsystem control block (SCB) to access said device, said SCB containing information affording access to said device; a read only memory for storing a SCSI advanced basic I/O system (ABIOS) program, said SCSI BIOS program being operable to control access to said device in response to receiving an BIOS request block, which includes said SCB; a main memory for storing a multitasking operating system, application program and data structures; and a processor for executing application programs and operating said data processing system under control of said operating system;  
 characterised in that the data processing system further comprising a device driver system comprising:  
 a SCSI device class driver (SDCD) stored in said main memory, said SDCD being callable by said operating system and operable in response to receiving a first request packet from said operating to construct a second request packet and said SCB;  
 and a SCSI generic device driver (SGDD) stored in said main memory, said SGDD being callable by said SDCD and operable upon receiving said second request packet from said SDCD to construct said BIOS request block and pass said BIOS request block to said SCSI BIOS, said second request packet containing information for passing said SCB to said SCSI BIOS.
2. A data processing system in accordance with claim 1, wherein:  
 said driver system is layered, said SDCD being between said operating system and said SGDD, said SGDD being between said SDCD and said SCSI BIOS.
3. A data processing system in accordance with claim 2 wherein:  
 said SDCD is operative to translate an operating system request in said first request packet into at least one SCSI device access command that is stored in said SCB.
4. A data processing system in accordance with claim 2 or 3 wherein:  
 said system includes plural SCSI devices of different SCSI device classes;  
 and said driver system includes plural SDCDs one for each of said SCSI device classes.
5. A data processing system in accordance with claim 4 wherein:  
 said plural SDCDs are in the same layer between said operating system and said SGDD, said operating system being operative to direct said first request packet to one of said SDCDs in accordance with said device class.
6. A data processing system in accordance with any one of the preceding claims wherein:  
 said SGDD is operable to sequentially construct said BIOS request block, call said SCSI BIOS, and wait for an interrupt or a timeout to occur, said SGDD being further operable to handle said interrupt and said timeout.
7. A data processing system in accordance with claim 6 wherein:  
 said data processing system includes a plurality of SCSI devices;  
 said operating system being operable to receive plural concurrent requests to access different ones of said SCSI devices;  
 and said SGDD being operable to construct plural BIOS request blocks one for each of said requests, and to form said plural BIOS request blocks into a started queue.
8. A data processing system in accordance with claim 7 wherein:  
 said SGDD, in response to a subsequent request to access a busy SCSI device, places an associated BIOS request block or such request in a waiting queue, said SGDD being operable upon receiving an interrupt when such busy device becomes accessible to remove said associated BIOS request block from said waiting queue and place it on said started queue.
9. A data processing system in accordance with claim 6 wherein:  
 said SGDD is operable upon the occurrence of an interrupt to determine if an error occurred while accessing a requested device and automatically furnish sense data to said SDCD if there was an error.

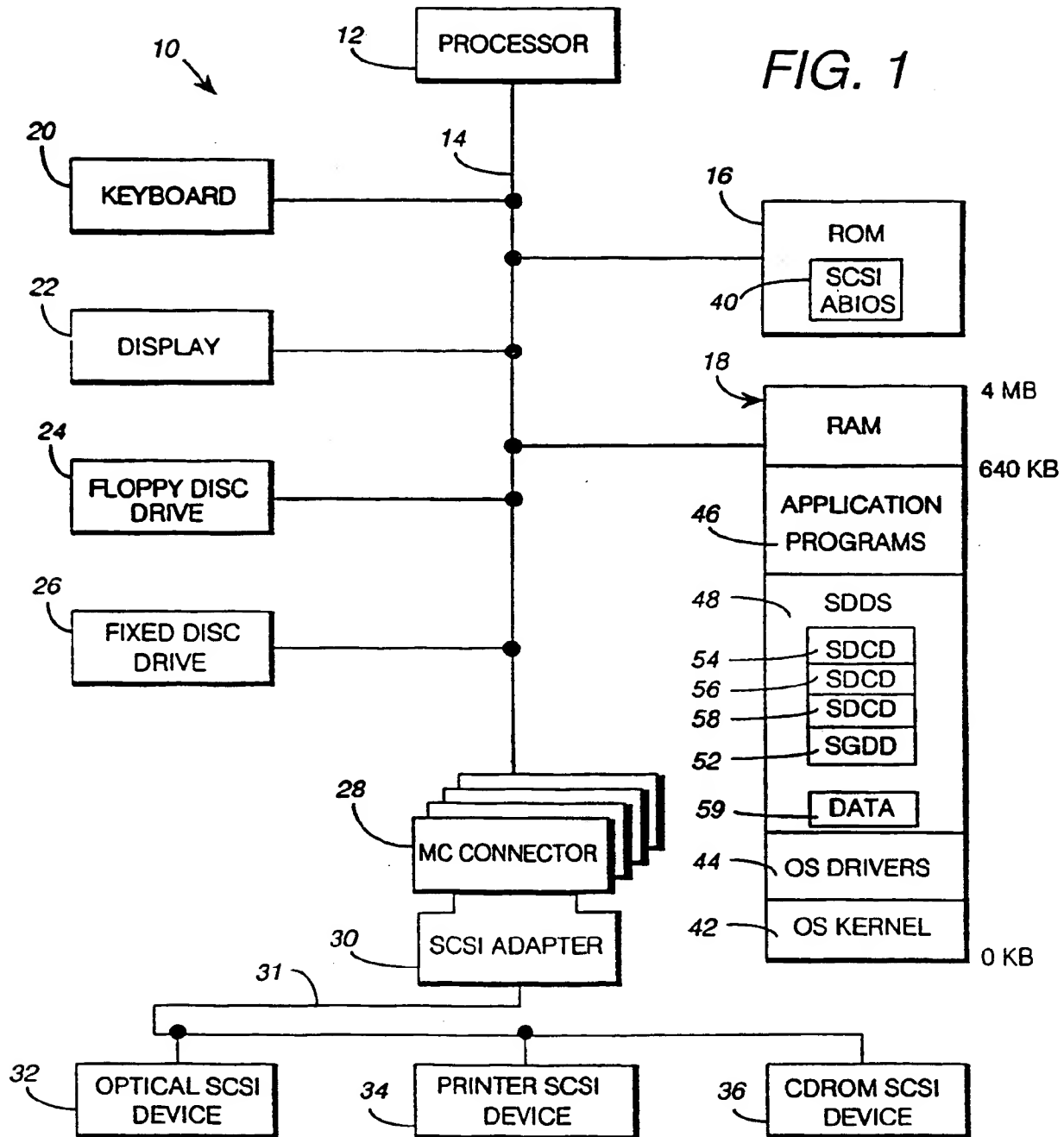




FIG. 2

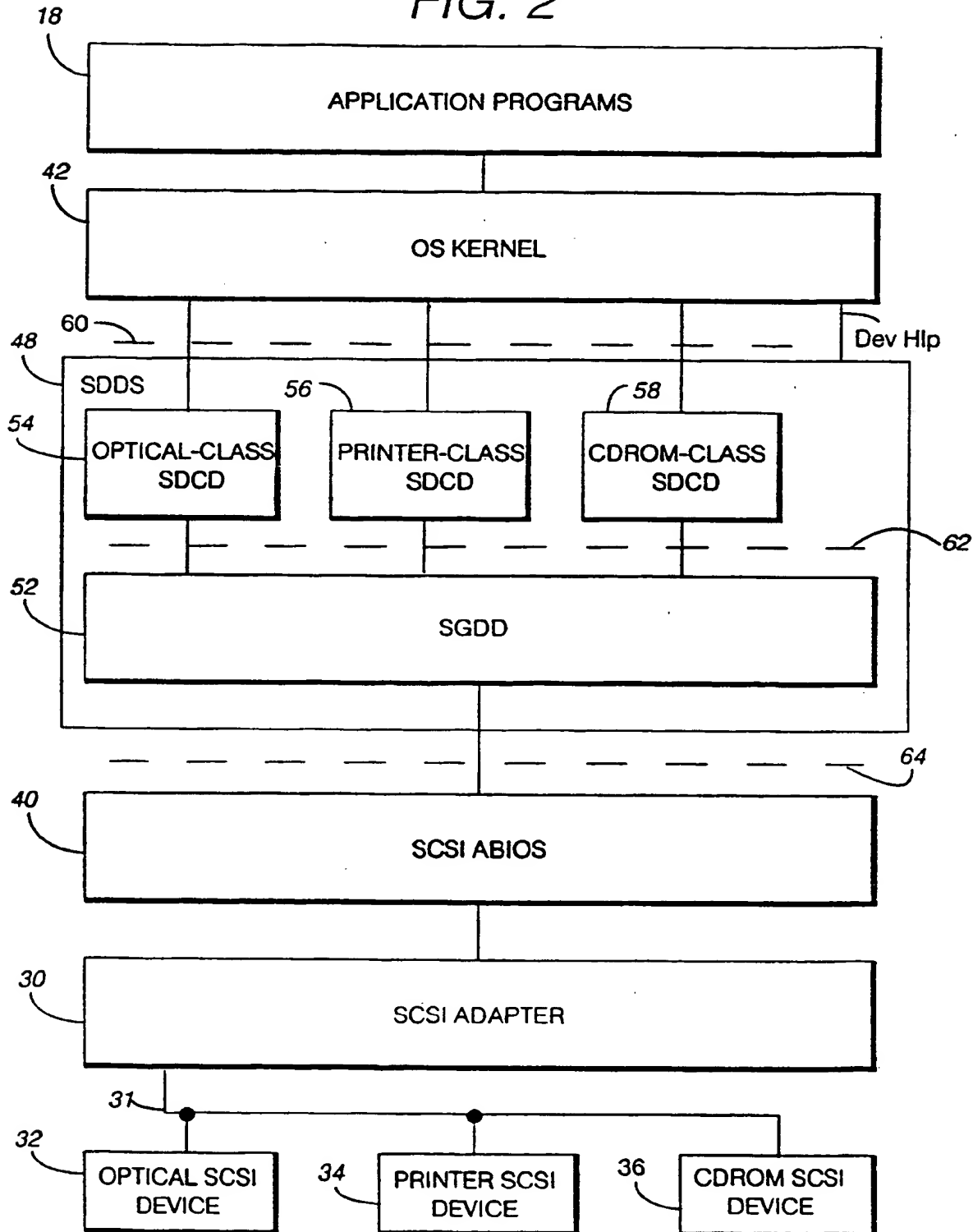


FIG. 3

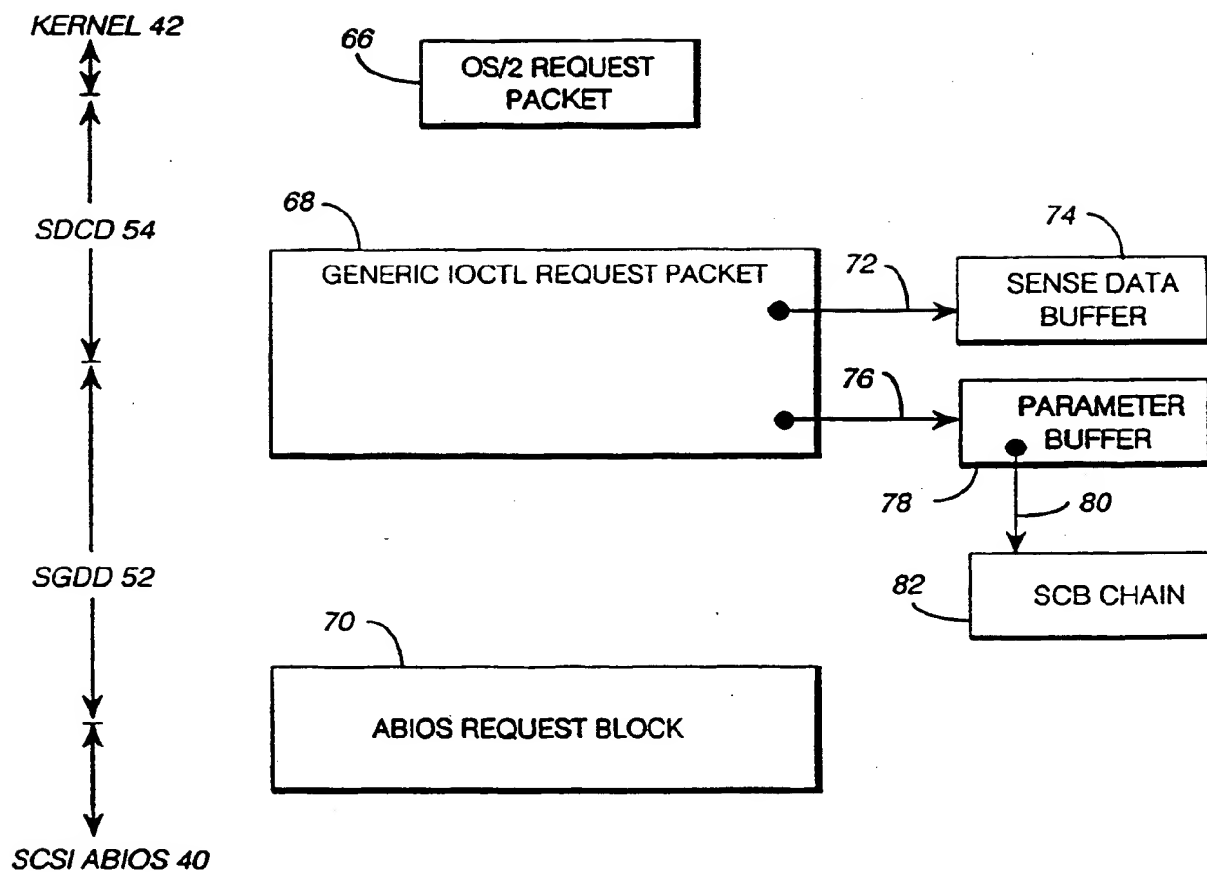


FIG. 4

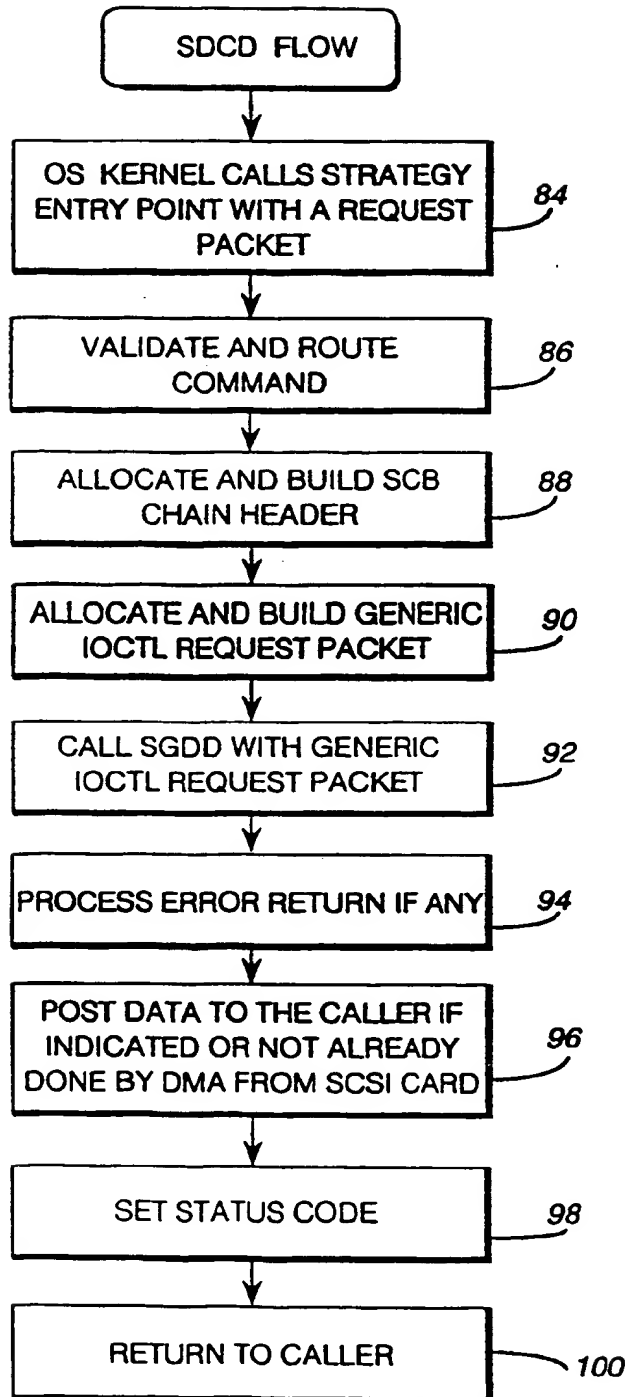


FIG. 5A

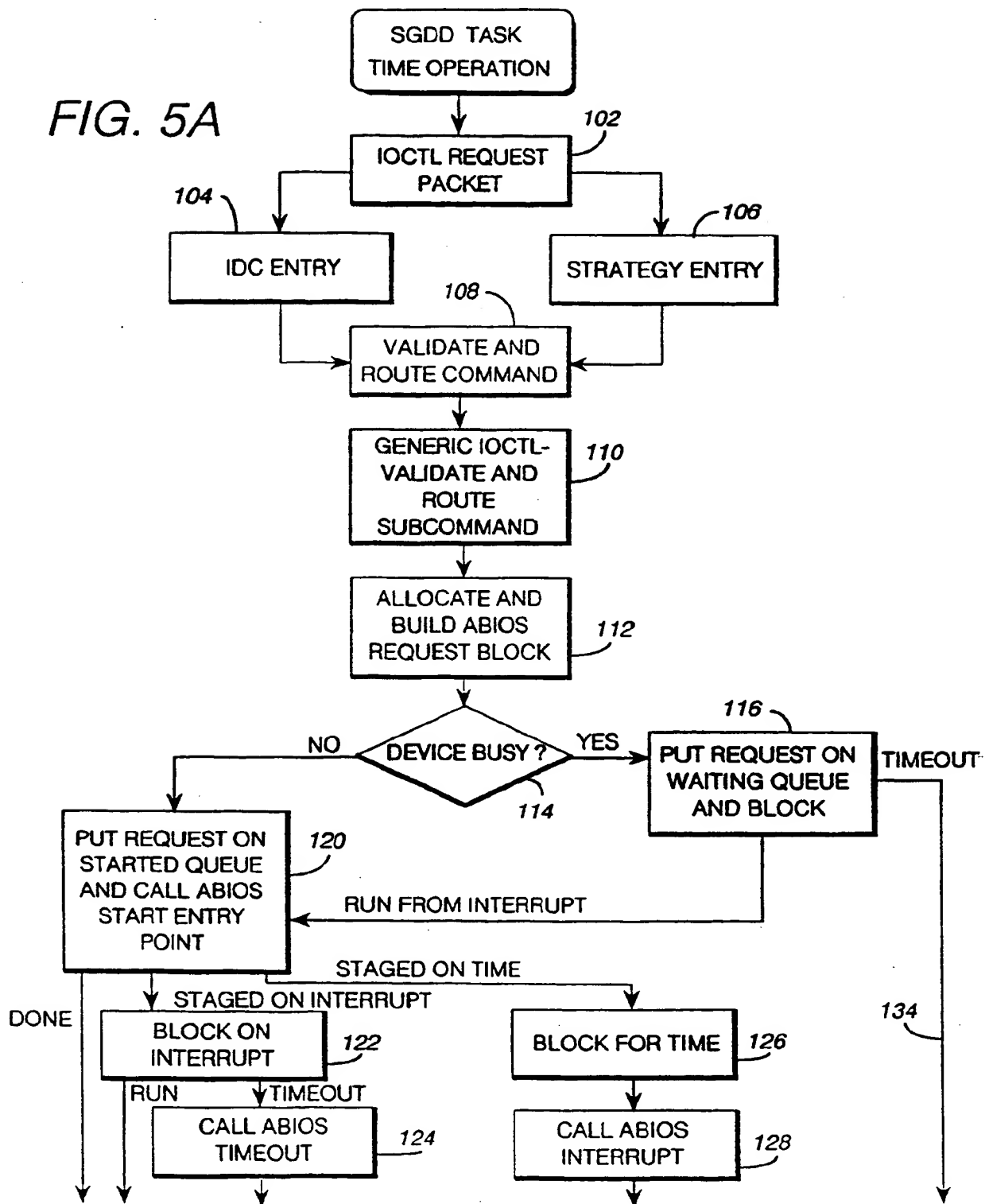


FIG. 5B

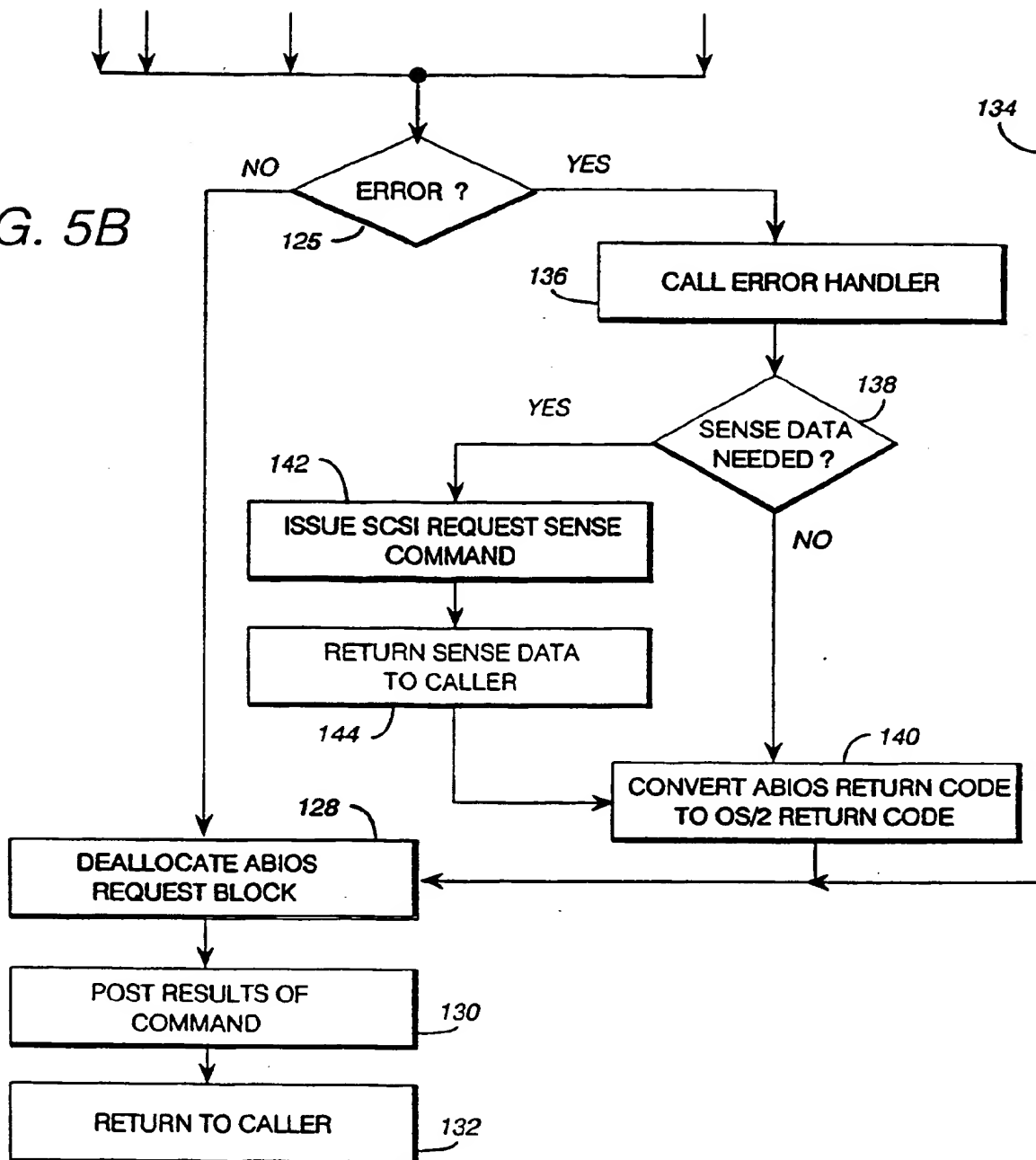
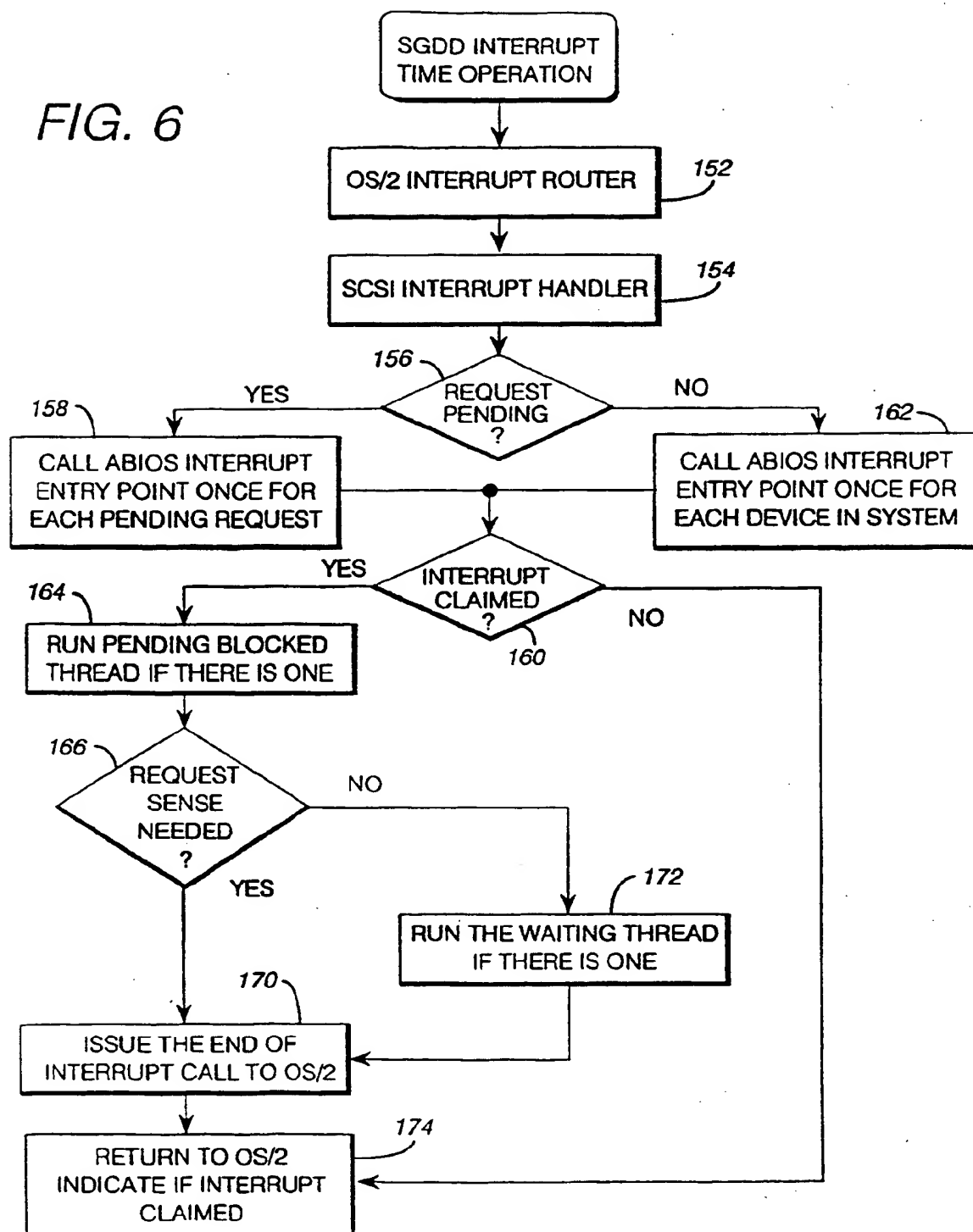


FIG. 6





European Patent  
Office

# EUROPEAN SEARCH REPORT

Application Number

EP 92 30 0919

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
Y	IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 15, no. 4, April 1989, NEW YORK US pages 437 - 447; D. DIAS ET AL: 'Integrated Concurrency-Coherency Controls for Multisystem Data Sharing'	1	G06F13/38
A	* page 438, left column * * page 439, right column - page 440, left column * * page 442, right column, paragraph A *	2-9	
Y	SYSTEMS INTEGRATION vol. 23, no. 7, July 1990, NEWTON, MA, US pages 61 - 70; DAVID SIMPSON: 'Your SCSI nightmare is over'	1	
A	* page 62, left column - page 63, right column * * page 64, right column *	2-4	
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 32, no. 2, July 1989, NEW YORK US pages 120 - 121; 'DFCI Master Slave Busy Queuing Algorithm' * page 120, paragraph 1; figure 1 *	1-9	
A	US-A-4 470 115 (WEHR) * column 1, line 10 - line 28 * * column 3, line 66 - column 4, line 19 *	1	
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 27 MAY 1992	Examiner NGUYEN Xuan Hiep C.
<p><b>CATEGORY OF CITED DOCUMENTS</b></p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons * : member of the same patent family, corresponding document</p>			

EPO FORM 1503 (3.92) (P0401)

This page Blank (1/15/01)